Lista zadań. Nr 6. 14 maja 2025

Algorytmy i Struktury Danych

IIUWr. II rok informatyki.

1. (2pkt) Podaj nierekurencyjną wersję procedury *Quicksort*, która
   - działa *w miejscu*, tj. poza tablicą z danymi (**int** $A[n]$) używa tylko stałej (niezależnej od n) liczby komórek typu **int** (zakładamy, że $\max(n, \max\{A[i] \mid i = 1,..,n\})$ jest największą liczbą jaką może pomieścić taka komórka),
   - czas jej działania jest co najwyżej o stały czynnik gorszy od czasu działania wersji rekurencyjnej.
2. (1pkt) Podaj algorytm sprawdzający izomorfizm drzew nieukorzenionych.
3. (1.5 pkt) Oszacuj oczekiwany czas działania Algorytmu Hoare'a (znajdowania mediany w ciągu). Mile widziane będzie zastosowanie metody Fredmana (z artykułu załączonego na stronie wykładu).
4. (2pkt) Niech $h(v)$ oznacza odległość wierzchołka v do najbliższego pustego wskaźnika w poddrzewie o korzeniu v. Rozważamy drzewa binarne, równoważone poprzez utrzymywanie następującego warunku:

   $$h(\text{lewy syn } v) \geqslant h(\text{prawy syn } v) \text{ dla każdego wierzchołka } v.$$

   Pokaż, w jaki sposób można na nich wykonywać operacje złączalnych kolejek priorytetowych (tj. wstawianie elementu, usuwanie minimum, łączenie drzew).
5. (0.5pkt) Udowodnij, że każde drzewo BST można przekształcić operacjami rotacji w dowolne inne drzewo BST .
6. (2pkt) Napisz procedurę $\mathit{Split}(T, k)$ rozdzielającą drzewo AVL T na dwa drzewa AVL: jedno zawierające klucze mniejsze od k i drugie zawierające pozostałe klucze. Jaka jest złożoność Twojej procedury?
7. (2pkt) Zaproponuj strukturę danych do pamiętania ciągu liczbowego i wykonywania na nim operacji:
   - $\mathit{insert}(i, a)$ - wstaw liczbę a na pozycję i w ciągu;
   - $\mathit{delete}(i)$ - usuń element znajdujący się na pozycji i;
   - $\mathit{find}(i)$ - podaj wartość, znajsującą się na pozycji i;
   - $\mathit{sum\text{-}of\text{-}even}()$ -podaj sumę elementów znajdujących się na pozycjach parzystych.

   Operacja $\mathit{insert}(i, a)$ przeształca ciąg $x_1, x_2, \ldots, x_n$ w ciąg $x_1, x_2, \ldots, x_{i-1}, a, x_i, x_{i+1}, \ldots, x_n$, a operacja $\mathit{delete}(i)$ przekształca ten ciąg w ciąg $x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$.
8. (1.5pkt) Bolesną dolegliwością związaną z drzewami AVL jest konieczność poświęcenia dwóch bitów w każdym węźle na pamiętanie współczynnika zrównoważenia. Zastanów się, czy aby na pewno mamy do czynienia z "koniecznością".
9. (**Z** 2pkt) Ułóż algorytm sortujący stabilnie i w miejscu ciągi rekordów o kluczach ze zbioru $\{1, 2, 3\}$.

Zadania dodatkowe - do samodzielnego rozwiązywania

1. (0pkt) Pokaż, że *Quicksort* działa w czasie $\Theta(n \log n)$, gdy wszystkie elementy tablicy A mają tę samą wartość.

2. (0pkt) Pokaż, że *Quicksort* działa w czasie $\Theta(n^2)$, gdy tablica A jest uporządkowana niemalejąco.

3. (0pkt) Załóżmy, że na każdym poziomie rekursji procedury *Quicksort* procedura *partition* dzieli daną tablicę na dwie podtablice w proporcji $1-\alpha$ do α, gdzie $0 < \alpha eq \frac{1}{2}$ jest stałą. Pokaż, że minimalna głębokość liścia w drzewie rekursji wynosi około $-\frac{\log n}{\log \alpha}$ a maksymalna głębokość liścia wynosi około $-\frac{\log n}{\log(1-\alpha)}$.

4. (1pkt) Opracuj wersję algorytmu *Quicksort*, która będzie efektywnie działać na ciągach zawierających wielokrotne powtórzenia kluczy.

5. (2pkt) Opracuj wersję algorytmu *Mergesort*, która działa w miejscu.

6. (1,5pkt) Zaproponuj strukturę danych do pamiętania zbioru liczbowego i wykonywania na nim operacji: *insert*, *delete*, *mindiff*. Ostatnia z tych operacji zwraca jako wynik najmniejszą różnicę między dwoma elementami zbioru.

7. (2pkt) Niech $A = a_1, a_2, \ldots, a_n$ będzie ciągiem elementów oraz niech p i q będą dodatnimi liczbami naturalnymi. Rozważmy p-podciągi ciągu A, tj. podciągi utworzone przez wybranie co p-tego elementu. Posortujmy osobno każdy z tych podciągów. Powtórzmy to postępowanie dla wszystkich $q-$podciągów. Udowodnij, że po tym wszystkie $p-$podciągi pozostaną posortowane.

8. (2pkt) n-elementowym ciągiem o jednym zaburzeniu nazywamy dowolny ciąg, który może być otrzymany z ciągu $\{1, 2, \ldots, n\}$ poprzez wykonanie jednej transpozycji. Załóżmy, że algorytm *InsertSort* będzie uruchamiany jedynie na ciągach o jednym zaburzeniu. Zbadaj średnią złożoność algorytmu przy założeniu, że dla każdego n, wszystkie takie ciągi n-elementowe są jednakowo prawdopodobne.

9. (1pkt) (Poprawność procedury *Partition*). Rozważ następującą procedurę:

```
Partition(A, p, r)
x ← A[p]
i ← p − 1
j ← r + 1
 while true  do
     repeat j − −
         until A[j] ⩽ x
     repeat i + +
         until A[i] ⩾ x
     if i < j
         then zamień A[i] ↔ A[j]
         else  return j
```

   Udowodnij co następuje

   (a) Indeksy i oraz j nigdy nie wskazują na element A poza przedziałem $[p..r]$.

   (b) Po zakończeniu *Partition* indeks j nie jest równy r (tak więc podział jest nietrywialny).

   (c) Po zakończeniu *Partition* każdy element $A[p..j]$ jest mniejszy lub równy od dowolnego elementu $A[j+1, r]$.

10. (1pkt) Ułóż algorytm sortujący ciąg n liczb całkowitych w czasie $O(n)$ i pamięci $O(n)$. Przyjmij, że liczby są z zakresu **long long**.

11. (2pkt) *Serią* w ciągu nazwiemy dowolny niemalejący podciąg kolejnych jego elementów. Seria jest *maksymalna*, jeśli nie można jej rozszerzyć o kolejne elementy. Załóżmy, że algorytm *InsertSort* uruchamiany będzie jedynie na permutacjach zbioru $\{1, 2, \ldots, n\}$, które można rozbić na co najwyżej dwie serie maksymalne. Zbadaj średnią złożoność algorytmu przy założeniu, że dla każdego n, wszystkie takie permutacje n-elementowe są jednakowo prawdopodobne.

12. (2pkt) Rozważmy permutacje liczb $\{1, 2, \ldots, n\}$, których wszystkie 2-podciągi i 3-podciągi są uporządkowane.

    (a) Ile jest takich permutacji?

    (b) Jaka jest maksymalna liczba inwersji w takiej permutacji?

    (c) Jaka jest łączna liczba inwersji w takich permutacjach?